Abstract
Presentation
Back to Agenda
Next Abstract

Parallel Performance of Pure MATLAB “M-files” versus “C-code” as applied to formation of Wide-Bandwidth and Wide-Beamwidth SAR Imagery

Dr. John Nehrbass[1] nehrbass@ee.eng.ohio-state.edu 937-904-5139,
Dr. Mehrdad Soumekh[2] msoum@eng.buffalo.edu 716 645-3115 x 2138,
Dr. Stan Ahalt[1] sca@ee.eng.ohio-state.edu 614-292-0068,
Dr. Ashok Krishnamurthy[1] akk@ee.eng.ohio-state.edu 614-292-5604,
Dr. Juan Carlos Chaves[1] jchaves@arl.army.mil 301-394-0408
[1] Department of Electrical Engineering, The Ohio State University, Columbus, Ohio 43210
[2] Deparment of Electrical Engineering, State University of New York at Buffalo,
332 Bonner Hall, Amherst, NY 14260

Applicable Topic Areas:

- Case Study Examples of High Performance Embedded Computing
- Performance Modeling and Simulation for Benchmarking Embedded Systems
- Software Architecture, Reusability, Scalability, and Standards

Abstract

Once an algorithm is stable and leaves the research and development stage, it can be optimized for particular hardware architectures and run in production mode. Most code written never gets to this advanced stage, however the need for speed may be critical for realistic simulations. Likewise, one may be impressed with the long run times of simulations, which run for weeks or longer while using hundreds of processors in parallel to provide the needed information. However, it is often overlooked that the resources required to place such problems into formats suitable for such lengthy simulations many span several years. Grid quality, a labor-intensive step, supercedes numerical convergence criteria for accurate RCS predictions. Similarly, it is rare that an author is able to publish after only one run on the first bug free version of their code.

To the extent that tools can allow rapid prototyping, increased value is achieved. For scientists in the area of Signal and Image processing, MATLAB is often the tool of choice. MATLAB allows extremely rapid prototyping and debugging of complicated studies with a minimal of computer science expertise. Frequently an idea is implemented in MATLAB code and tested on small data sets. These small data sets provide outputs in sufficient time such that performance studies can be conducted. However, as the studies progress and the same codes are used on much larger real data sets, run times may grow to unrealistic lengths. This may force the research scientist to use another implementation language, such as C and MPI, and implement the code on parallel architectures. This code conversion is undesirable, time consuming, and error prone.

An alternative and convenient way to accomplish run time reduction is to use the MatlabMPI suite written by Jeremy Kepner of MIT. This suite of pure MATLAB code allows one to simulate many of the MPI functions within MATLAB. It accomplishes inter-processor communication via disk I/O and a common disk drive. MATLAB software requires a license per computational node. Thus if one runs MatlabMPI on a shared memory machine, only one MATLAB license need be used for any given implementation. When run a distributed system, such as the IBM SP2/3 or a set of Linux clusters, then a license for each grouping of machines is required. For large jobs, (i.e. 256 processors or more) the additional costs of licenses may not be justified. Realizing the combined value of MATLAB, MatlabMPI, the High Performance Computational Modernization Program (HPCMP), and distributed architectures, an alternative is presented. MatlabMPI works by creating scripts that contain instructions for starting individual MATLAB processes, one for each processor desired on a given node. Executable code can be substituted within the scripts wherever a MATLAB process was identified. The MATLAB compiler toolbox may be used to create MATLAB executable code (MEX) fused within a C wrapper. The C wrapper simply allows one to easily define the rank of a launched process and pass this information to the MEX code.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **20 AUG 2004**	2. REPORT TYPE **N/A**	3. DATES COVERED **-**

4. TITLE AND SUBTITLE
Parallel Performance of Pure MATLAB M-files versus C-code as applied to formation of Wide-Bandwidth and Wide-Beamwidth SAR Imagery

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Department of Electrical Engineering, The Ohio State University, Columbus, Ohio 43210; Deparment of Electrical Engineering, State University of New York at Buffalo, 332 Bonner Hall, Amherst, NY 14260

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR’S ACRONYM(S)

11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM001694, HPEC-6-Vol 1 ESC-TR-2003-081; High Performance Embedded Computing (HPEC) Workshop (7th)., The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**50**	

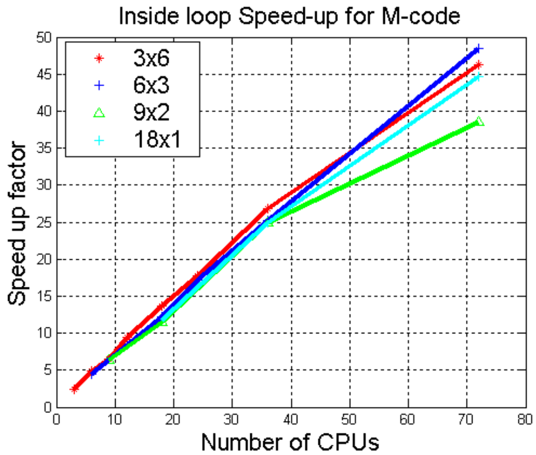
Abstract Presentation Back to Agenda Next Abstract

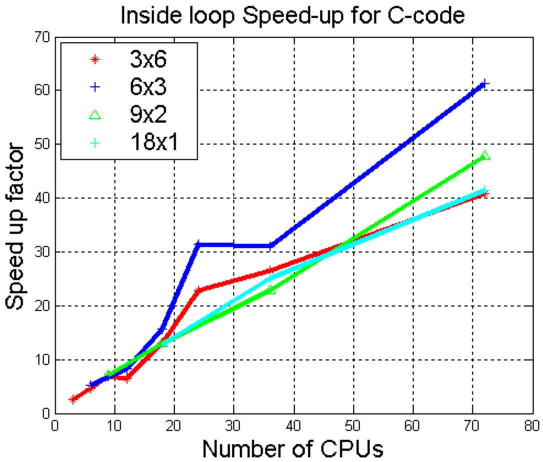
These altered scripts no longer require any MATALB licenses at run time. Thus distributed systems can now affordably be added to the suit of resources that MatlabMPI can be used on. This presentation will illustrate the comparison of timing between running pure parallel MATLAB code with the automatically created C compiled version of the same code. A sample speed up comparison is illustrated below.

As rapid prototyping is the main focus of this effort, the illustration is implemented on a fairly complicated image formation code that generates large SAR images. Initial development was conducted on smaller sets of simulated data and thus relatively small images were produced. However, the real applications needed to generate images with dimensions of 4 by 7 kilometers at resolution of 3 meters, which generates images of dimensions 1333 pixels by 2333 pixels. The phase history processed consisted of over 18,000 slow time samples each containing thousands of frequencies or fast time complex data samples. When processing real data on 2Ghz Pentium IV Pc machines, run times spanned from over 1 hour to approximately 2 hours. The variations in run times were caused by variations in optimizing parameters and availability of RAM. After a very modest effort of porting this code to HPC resources and using parallel processors to run the same parameter sets, run times were reduced to less than 2 minutes. It is believed that with a finer grain parallization, run times will be reduced to less than 15 seconds and thus achieve real time processing claims. These results, images, timings, and details will be presented at the HPEC conference.

The formation of general wide-bandwidth and wide-beamwidth SAR imagery follows an approach that is based on the SAR ***wavefront*** theory. The SAR wavefront theory was introduced in the past decade for approximation-free image formation. Meanwhile, the fundamental SAR signal properties that are established via this theory have been proven to be useful not only for image formation but also for addressing pre- and post-processing problems that are encountered in practical SAR systems. These include RFI suppression, suppression of azimuthal spatial aliasing in wide-beamwidth SAR systems, motion compensation and auto-calibration, geo-registration and calibration of multipass data for change detection. The specific issue that is examined in this study is the large integration angle of these SAR systems; in this case, the user faces processing relatively large FFTs for Doppler processing to avoid azimuthal spatial aliasing. A process that we refer to as ***subaperture-based digital spotlighting*** is used to circumvent the above-mentioned problem; this also makes the implementation ideal in a multi-processor environment.

In order to facilitate a more efficient I/O the phase history data is broken up in to many files (subapertures), each containing only hundreds of subsets of the total slow time data sets. Processing through each of the files is done in an outer most loop. Auto focusing through one file is independent of processing through another, and thus this technique is an excellent candidate for parallel implementation. If 1,2,3,6,9, or 18 processors are used, the outer loop can be perfectly load balanced. Likewise, an inner loop independently performs an FFT over sub-apertures. This loop is also optimal for parallization. For this data set 72 sub-apertures are possible and thus the code was scaled from 1 to 72 processors.





This publication was made possible through support provided by DoD HPCMP PET activities through Mississippi State University under the terms of Agreement No. #GS04T01BFC0060. The opinions expressed herein are those of the author(s) and do not necessarily reflect the views of the DoD or Mississippi State University.

Abstract
Presentation
Back to Agenda
Next Abstract

Parallel Performance of Pure MATLAB "M-files" versus "C-code“ as applied to formation of Wide-Bandwidth and Wide-Beamwidth SAR Imagery

Nehrbass@ee.eng.ohio-state.edu

Parallel Performance of Pure MATLAB "M-files" versus "C-code" as applied to formation of Wide-Bandwidth and Wide-Beamwidth SAR Imagery

Dr. John Nehrbass[1]
Dr. Mehrdad Soumekh[2]
Dr. Stan Ahalt[1]
Dr. Ashok Krishnamurthy[1]
Dr. Juan Carlos Chaves[1]

[1] Department of Electrical Engineering, The Ohio State University, Columbus, Ohio 43210

[2] Deparment of Electrical Engineering, State University of New York at Buffalo,
332 Bonner Hall, Amherst, NY 14260

Outline

- MatlabMPI Overview
- Possible modifications/customizations
- SAR Imagery
- Parallel Performance “M” vs “C”
- Future Work and activities

MatlabMPI Overview

References:

- **http://www.mathworks.com/**
- The latest MatlabMPI information, downloads, documentation, and information may be obtained from

http://www.ll.mit.edu/MatlabMPI

MPI & MATLAB

- Message Passing Interface (MPI):
  - A message-passing library specification
  - Specific libraries available for almost every kind of HPC platform: shared memory SMPs, clusters, NOWs, Linux, Windows
  - Fortran, C, C++ bindings
  - Widely accepted standard for parallel computing.
- MATLAB:
  - Integrated computation, visualization, programming, and programming environment.
  - Easy matrix based notation, many toolboxes, etc
  - Used extensively for technical and scientific computing
  - Currently: mostly SERIAL code

What is MatlabMPI?

- It is a MATLAB implementation of the MPI standards that allows any MATLAB program to exploit multiple processors.
- It implements, the basic MPI functions that are the core of the MPI point-to-point communications with extensions to other MPI functions. (Growing)
- MATLAB *look and feel* on top of standard MATLAB file I/O.
- Pure M-file implementation: about 100 lines of MATLAB code.
- It runs anywhere MATLAB runs.
- Principal developer: Dr. Jeremy Kepner (MIT Lincoln Laboratory)

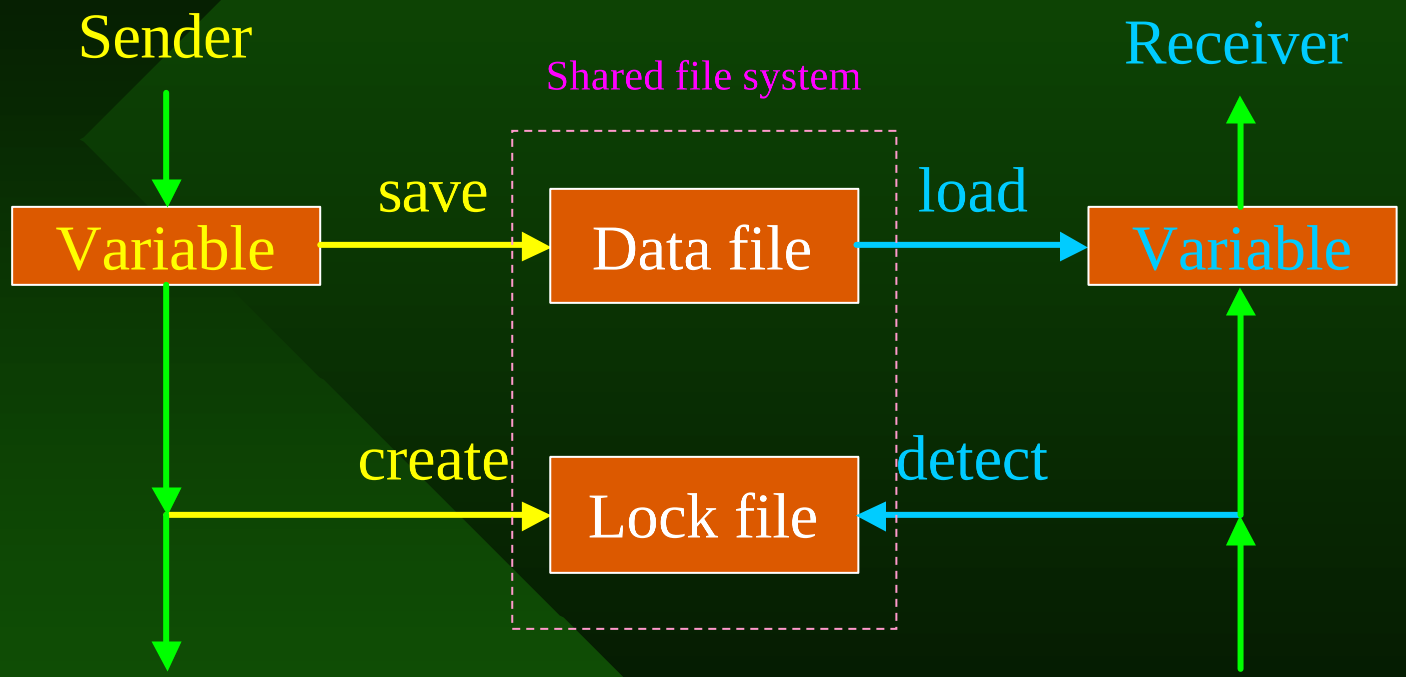
General Requirements

- As MatlabMPI uses file I/O for communication, a common file system must be visible to every machine/processor.
- On shared memory platforms: single MATLAB license is enough since any user is allowed to launch many MATLAB sessions.
- On distributed memory platforms: one MATLAB license per machine / node.
- Currently Unix based platforms only, but Windows support coming soon.

Basic Concepts

- Basic Communication:
  - Messages: MATLAB variables transferred from one processor to another
  - One processor sends the data, another receives the data
  - Synchronous transfer: call does not return until the message is sent or received
  - SPMD model: usually MatlabMPI programs are parallel SPMD programs. The same program is running on different processors/data.

Communication architecture



- Receiver waits until it detects the existence of the lock file.
- Receiver deletes the data and lock file, after it loads the variable from the data file.

Possible modifications/customizations

- ssh vs rsh
- Path variables
- System dependent information required to run MATLAB.

Master scripts

MatlabMPI creates 2 sets of scripts

Unix_Commands.sh – master script
This contains instructions for launching scripts on each desired node.

Unix_Commands.node_alias.sh –
This contains instructions for “what” is to be run on node_alias.

Unix_Commands.ssh example

```
ssh hpc11-1 –n ‘cd
   /work1/nehrbass/D_6_3x6; /bin/sh
   ./MatMPI/Unix_Commands.hpc11-1.0.sh
   &’ &

ssh hpc11-3 –n ‘cd
   /work1/nehrbass/D_6_3x6; /bin/sh
   ./MatMPI/Unix_Commands.hpc11-3.0.sh
   &’ &
```

Unix_Commands.hpc11-3.0.ssh example NCPU=6

```
matlab –display null –nojvm –nosplash < myprog.m > MatMPI/myprog.5.out &
touch MatMPI/pid.hpc11-3.$!
matlab –display null –nojvm –nosplash < myprog.m > MatMPI/myprog.4.out &
touch MatMPI/pid.hpc11-3.$!
matlab –display null –nojvm –nosplash < myprog.m > MatMPI/myprog.3.out &
touch MatMPI/pid.hpc11-3.$!
matlab –display null –nojvm –nosplash < myprog.m > MatMPI/myprog.2.out &
touch MatMPI/pid.hpc11-3.$!
matlab –display null –nojvm –nosplash < myprog.m > MatMPI/myprog.1.out &
touch MatMPI/pid.hpc11-3.$!
matlab –display null –nojvm –nosplash < myprog.m > MatMPI/myprog.0.out &
touch MatMPI/pid.hpc11-3.$!

# Possibly add code to prevent early batch termination
```

Batch termination problem

Problem: Master script finishes before all the spawned processes are done and thus the batch job terminates prematurely.

Solution: Add the following code at the end of each node_alias script

```
mystat=`ls MatMPI |grep Finished.done|wc –l`
while [[ $mystat –lt 1 ]];
do
   mystat=`ls MatMPI |grep Finished.done|wc –l`
   sleep 15;
done
```

Batch termination problem

Solution: create a file called "Finished.done" at any place in the MatlabMPI code when code termination is desired.

This file can be created when ever the desired global answer is available, however; it is strongly suggested that a clean termination, (i.e. all processes are finished), be implemented.

Executable Implementation

Problem: There are insufficient licenses when running on a distributed system of n nodes.
(Recall – each node requires a valid license.)

Solution: Convert the working part of the MATLAB code to an executable that can run without a license requirement and modify the existing scripts.

Implementation Steps

1.) Modify the Matlab MPI code so that the scripts are automatically modified to run an executable code.

2.) Create an executable code from “M-file” scripts.

3.) Run MatlabMPI to generate all the required scripts automatically.

4.) Submit a batch job to start the scripts.

Script changes – 1

Change the script from

```
matlab –display null –nojvm –nosplash < myprog.m >
MatMPI/myprog.5.out &
touch MatMPI/pid.hpc11-3.$!
:
```

To

```
myprog.exe 5 > MatMPI/myprog.5.out &
touch MatMPI/pid.hpc11-3.$!
:
```

MatlabMPI Changes – 1

This is most easily done by editing the file MatMPI_Commands.m

Change the line

From

```
matlab_command = [matlab_command ‘ < ‘ defsfile ‘ > ’  outfile ];
```

To

```
matlab_command = [‘myprog.exe ‘ num2str(rank) ‘ > ‘ outfile ];
```

Create an Executable – 2

Change the “M-file” to a function and add the 3 lines of code below.

```
function dummy=myprogf(my_cpu)

    % The function myprogf.m was created by converting the
    % MATLAB “M-file” script myprog.m to this function.
    %
    % Other comments
    %

    % Create MatlabMPI setup commands
    global MPI_COMM_WORLD;
    load MatMPI/MPI_COMM_WORLD;
    MPI_COMM_WORLD.rank = my_cpu;

    % The rest of the code myprog.m is appended without change
```

Executable wrapper – 2

```
#include <stdio.h>
#include <string.h>
#include “matlab.h”
#include “multpkg.h”

int main( int argc, char *argv[])    /* Used to call mlfMyprogf  */
{
  mxArray *my_cpu;
  int rank;
  rank=atoi(argv[1]);
  multpkgInitialize();
  my_cpu=mlfScalar(rank);
  mlfMyprogf(my_cpu);
  multpkgTerminate();
  return(0);
}
```

Generate All Scripts – 3

- Begin Matlab
- Add the path of MatlabMPI src *(ie addpath ~/MatlabMPI/src)*
  Hint: If the code is having problems seeing the src directory, either copy the src files to a local directory, or add the above line inside the source code.
- *Add a machine list as desired*
  (ie machines={};)
- *Run the MatlabMPI code to generate the required scripts.*
  (ie eval(MPI_Run('myprogf',64,machines));)

Generate All Scripts – 3

- Note that this will automatically launch the codes and scripts and thus this will run interactively.
- To save all scripts and submit via batch edit the MPI-Run.m function.

  Comment out the last two lines of this function as

  % unix([‘/bin/sh ‘ unix_launch_file]);
  % delete(unix_launch_file);

- This prevents the code from running from within MATLAB and also saves all the scripts generated by MatlabMPI.

Submit to batch – 4

- Dilemma:

  MatlabMPI generates scripts specific to a list of machines (nodes).

  Batch only provides machine information when execution starts.
- It is therefore possible to generate a set of scripts that are not matched to the resources available at run time.
- A solution to this problem is given on the next few slides.

Submit to batch – 4

- Place inside a batch script to create the file mat_run.m

```
echo “[s,w]=unix(‘hostname’);” > mat_run.m
echo “is(s==0)” >> mat_run.m
echo “   machines{1}=w(1:end-1)” >> mat_run.m
echo “   eval(MPI_Run(‘myprogf’,${NCPU},machines));” >> mat_run.m
echo “end” >> mat_run.m
echo “exit” >> mat_run.m
```

- Run the file “mat_run.m” in MATLAB and capture the output

```
matlab –nojvm –nosplash < mat_run.m >& mat_run.out
```

Recall that this only created the required scripts

Submit to batch – 4

Run the master script on the correct node

```
If ($UNAME == “hpc11-0”) then
   /bin/sh ./MatMPI/Unix_Commands.hpc11-0.0.sh
endif

If ($UNAME == “hpc11-1”) then
   /bin/sh ./MatMPI/Unix_Commands.hpc11-1.0.sh
endif

If ($UNAME == “hpc11-2”) then
   /bin/sh ./MatMPI/Unix_Commands.hpc11-2.0.sh
endif

If ($UNAME == “hpc11-3”) then
   /bin/sh ./MatMPI/Unix_Commands.hpc11-3.0.sh
endif
```

MatlabMPI ssh & rsh

- Some UNIX systems require ssh over rsh.
- To avoid problems with having to enter username / password pairs when launching a script from a master system to be run on other systems do the following:
- Step 1. Generate new key pairs:

ssh-keygen –t dsa

Hit enter when prompted for a pass phrase

MatlabMPI ssh & rsh

- ***This creates a public private key pair located in the .ssh directory. The public key*** *(id_dsa.pub)* ***needs to be copied to a common location***

- ***Step 2***
  ```
  cd ~/.ssh
  cat id_dsa.pub >> ~/.ssh/authorized_keys2
  chmod 644 ~/.ssh/authorized_keys2
  ```

- For more information please visit http://www.bluegun.com/Software/ssh-auth.html

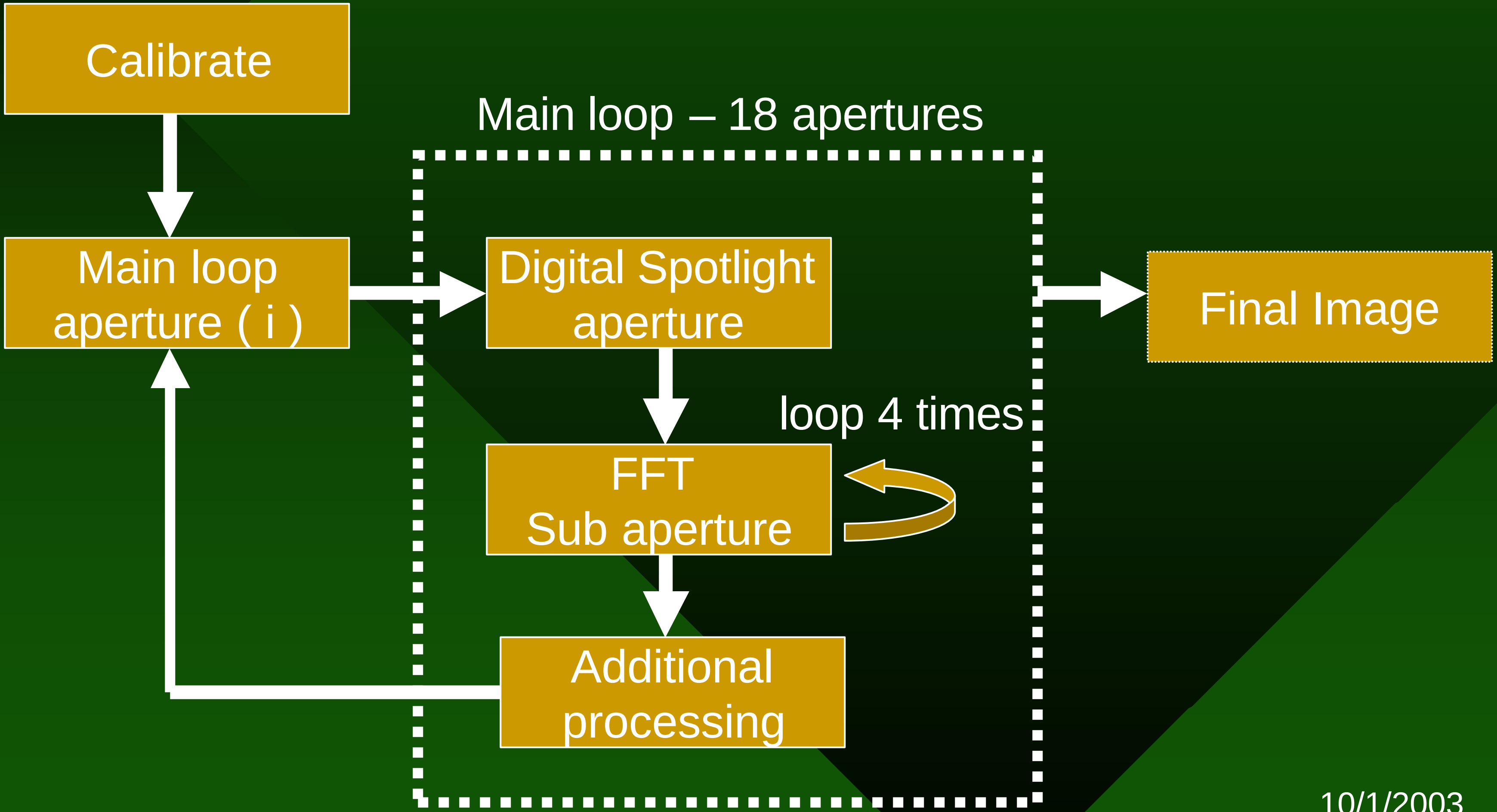
MatlabMPI ssh & rsh

- The HPCMP resources use kerberos. http://kirby.hpcmp.hpc.mil/
- When forwarding of credentials is used, one may be able to launch scripts on remote systems without implementing the previous steps.
- On ASC and ARL systems, ssh is sufficient.

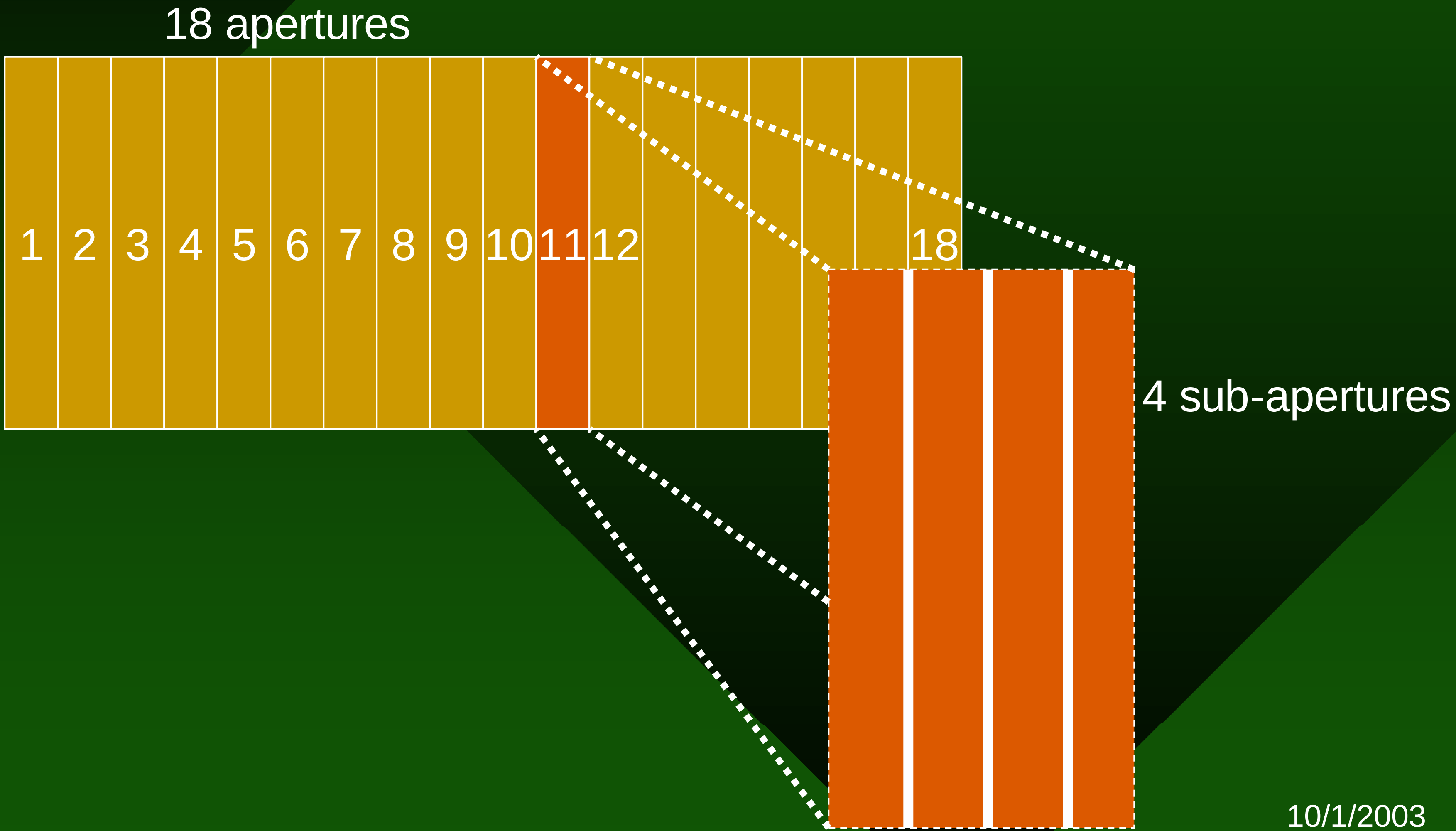
SAR Imagery

- Very large phase history data set are subdivided into 18 files (apertures).
- Auto focus through each file – independent of other files.
- Inner loop breaks aperture into 4 sub-apertures and performs FFT over each. Signal processing intense.

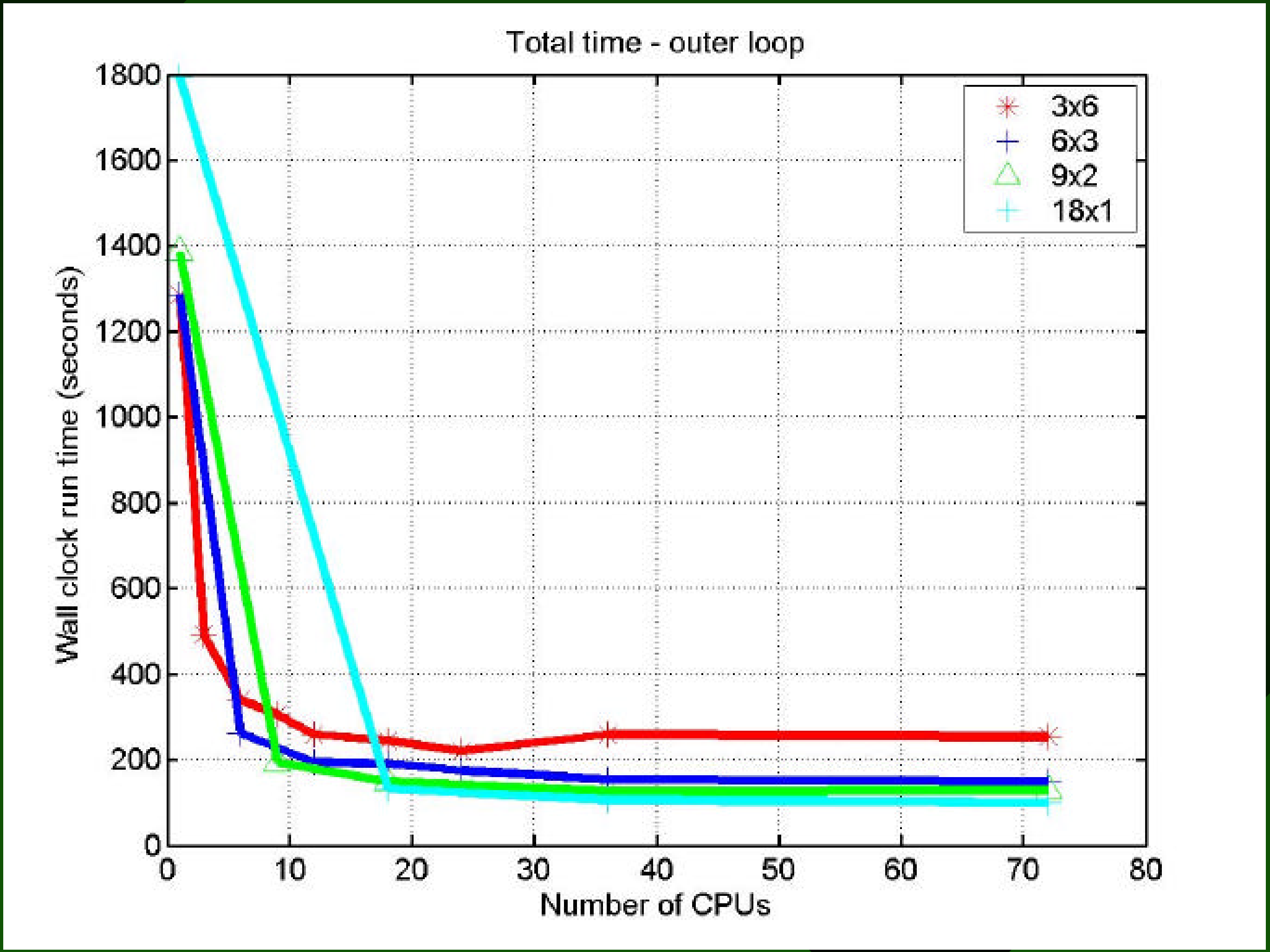
SAR Imagery


Calibrate
Main loop – 18 apertures
Main loop aperture (i)
Digital Spotlight aperture
Final Image
loop 4 times
FFT Sub aperture
Additional processing

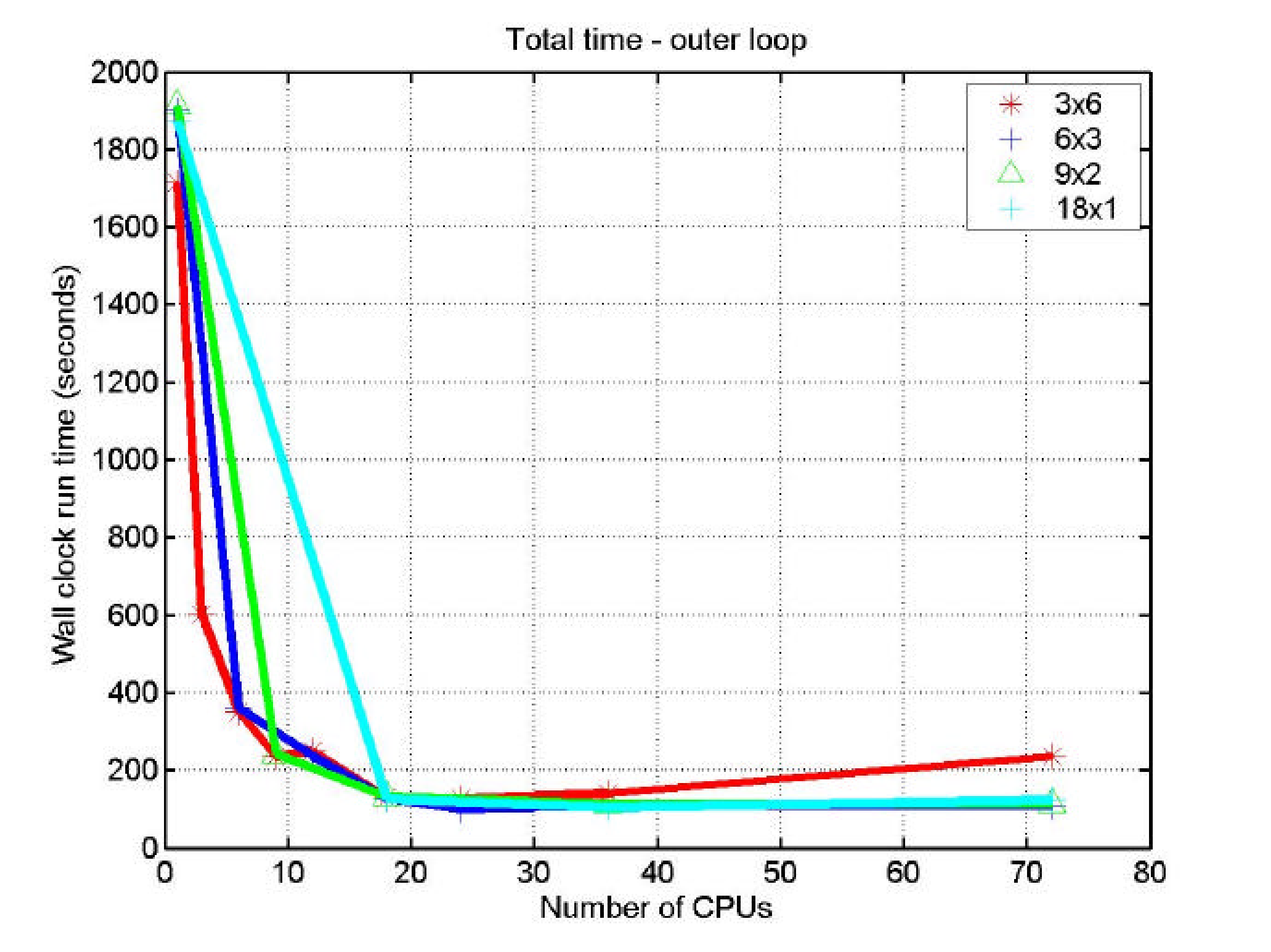
SAR Imagery



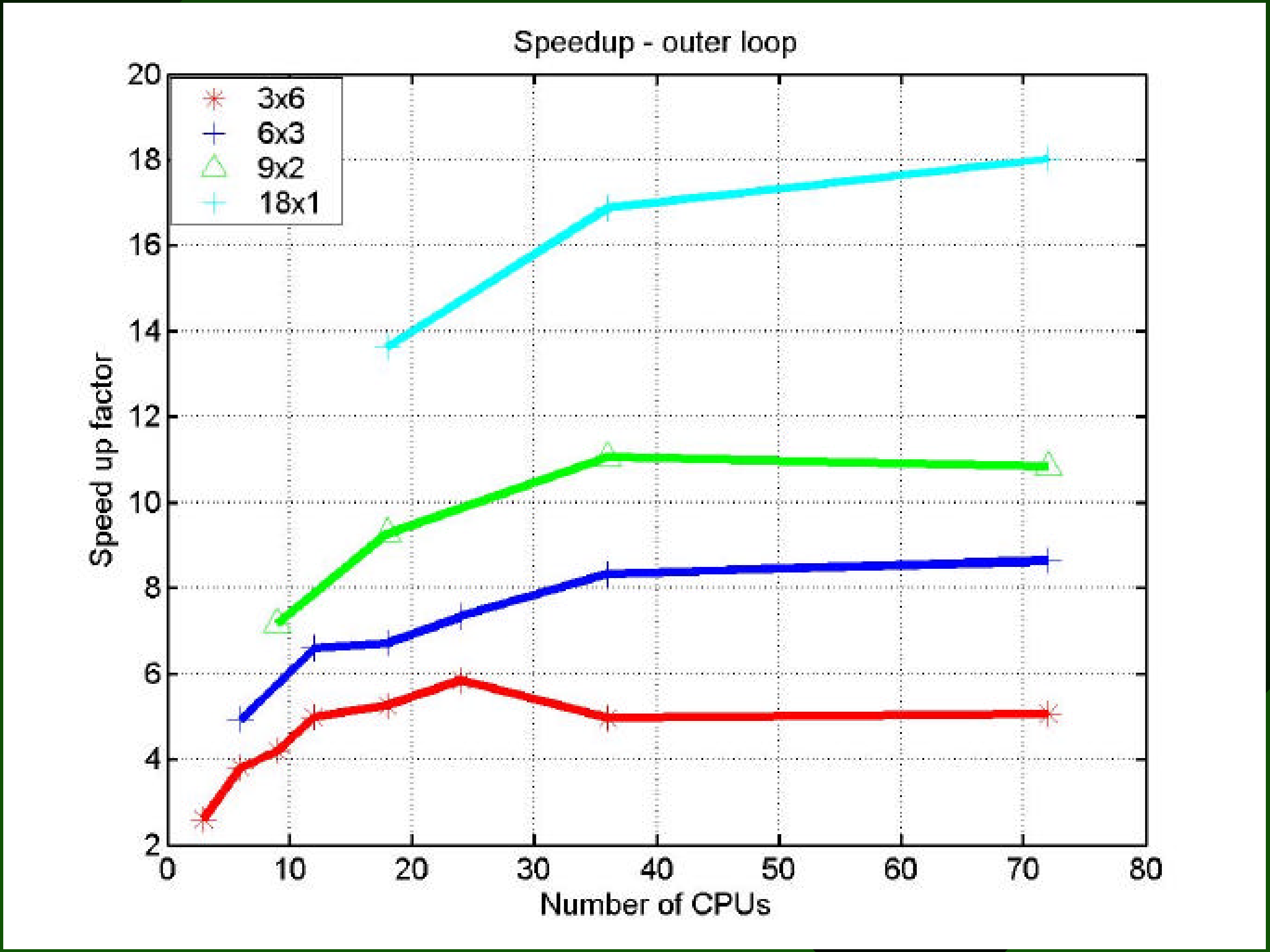
Parallel Performance “M” vs “C” Total time “M code”



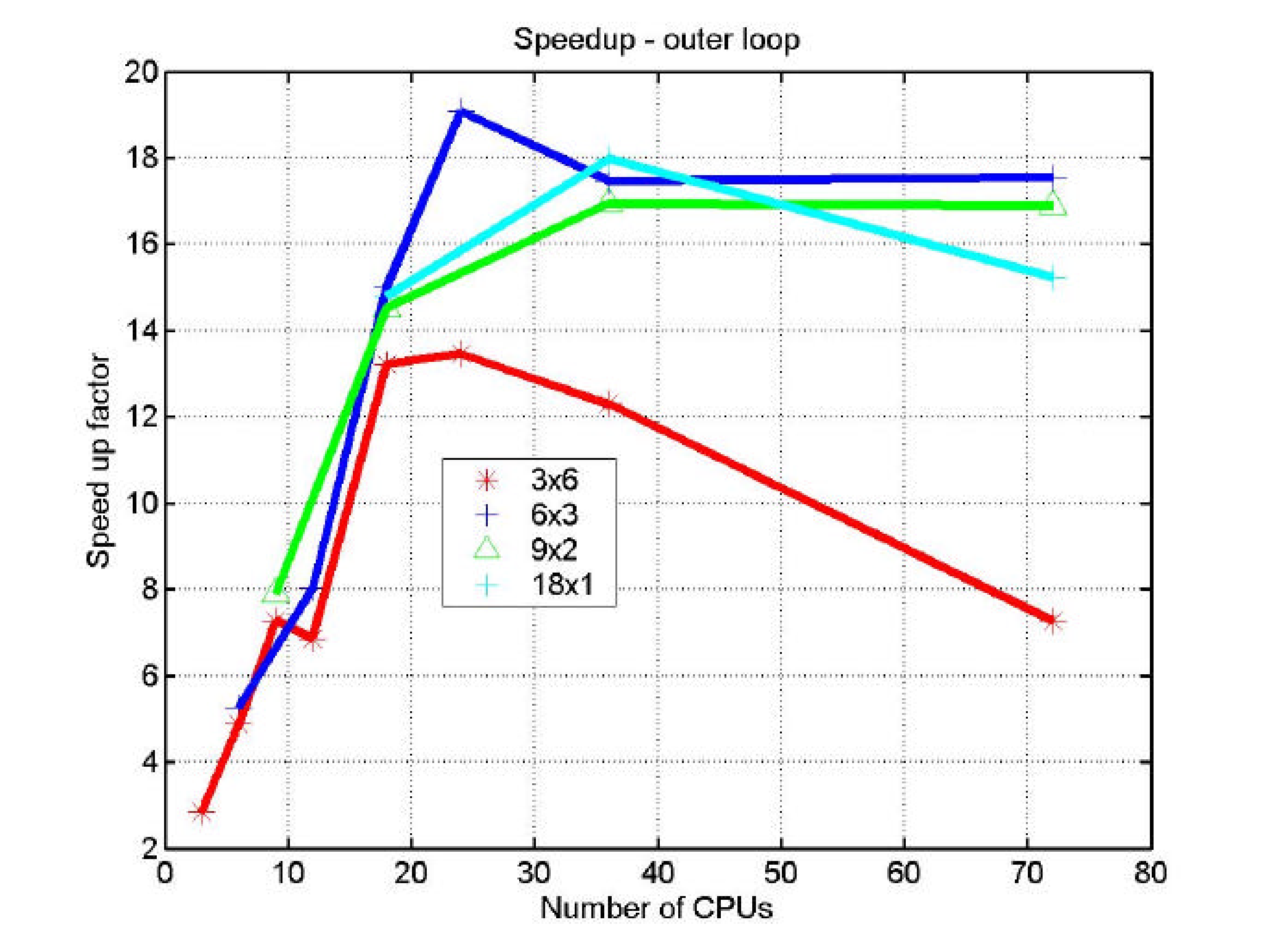
Total time “C code”



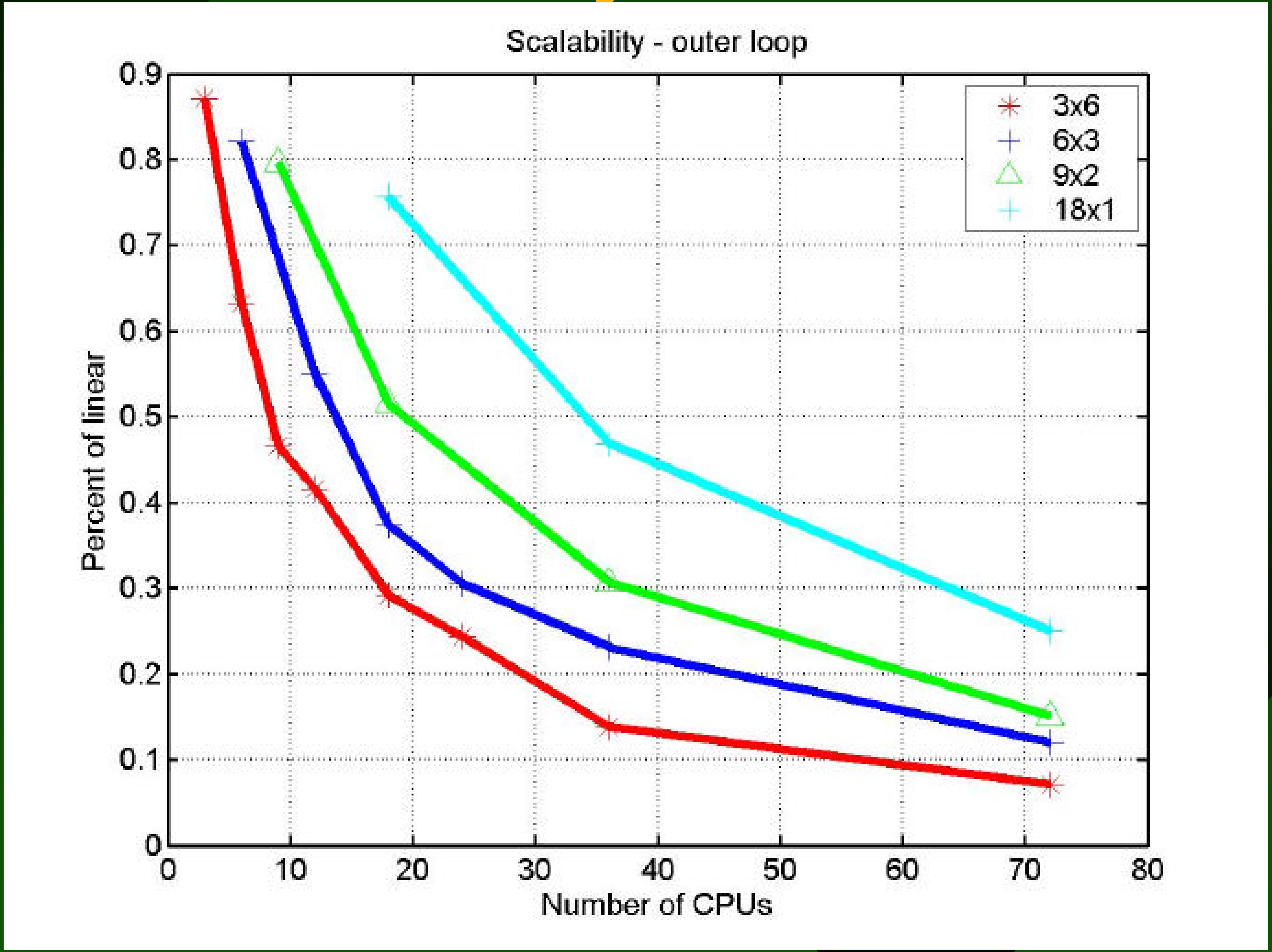
Speed up - outer loop “M code”



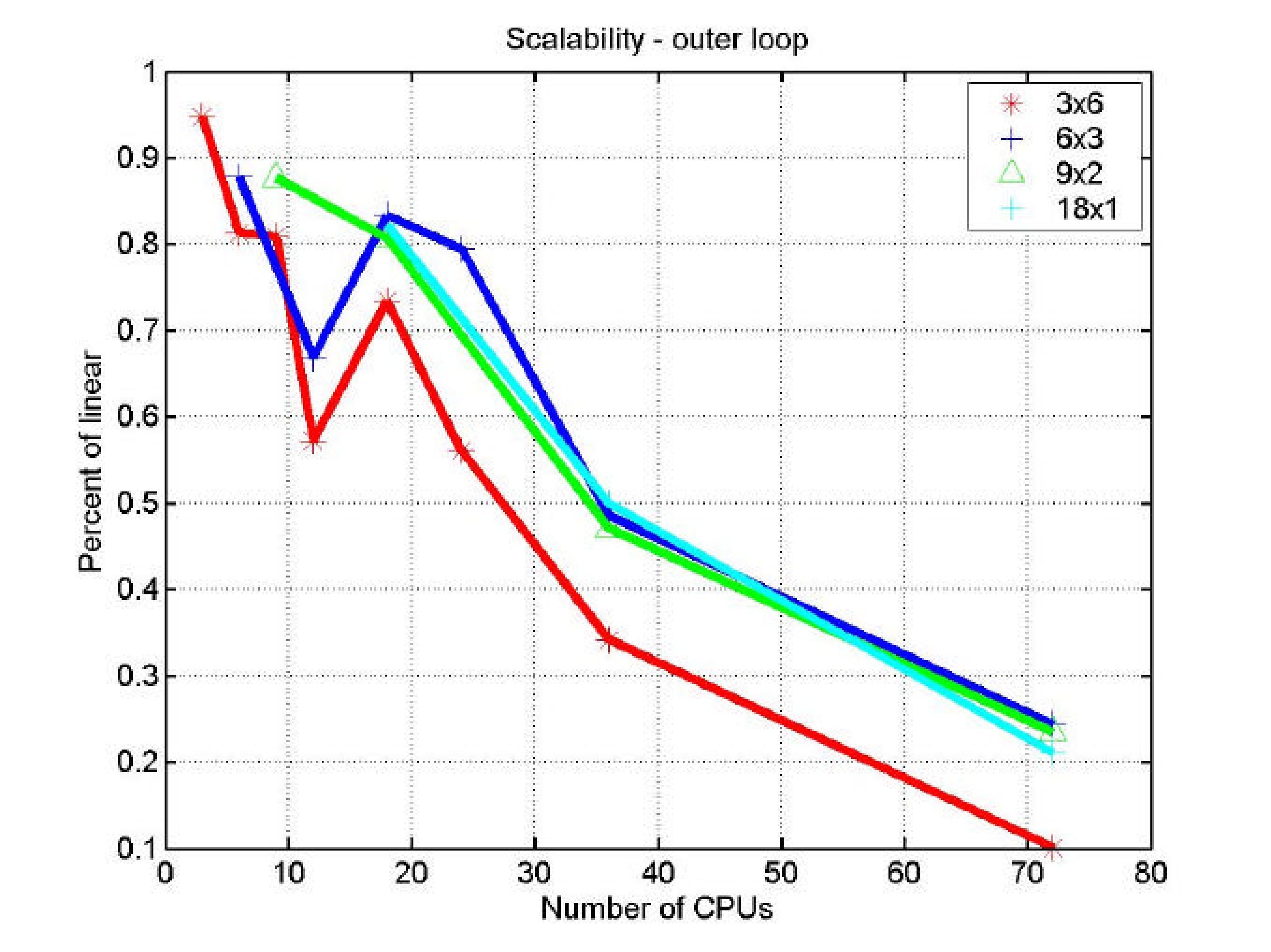
Speed up - outer loop “C code”



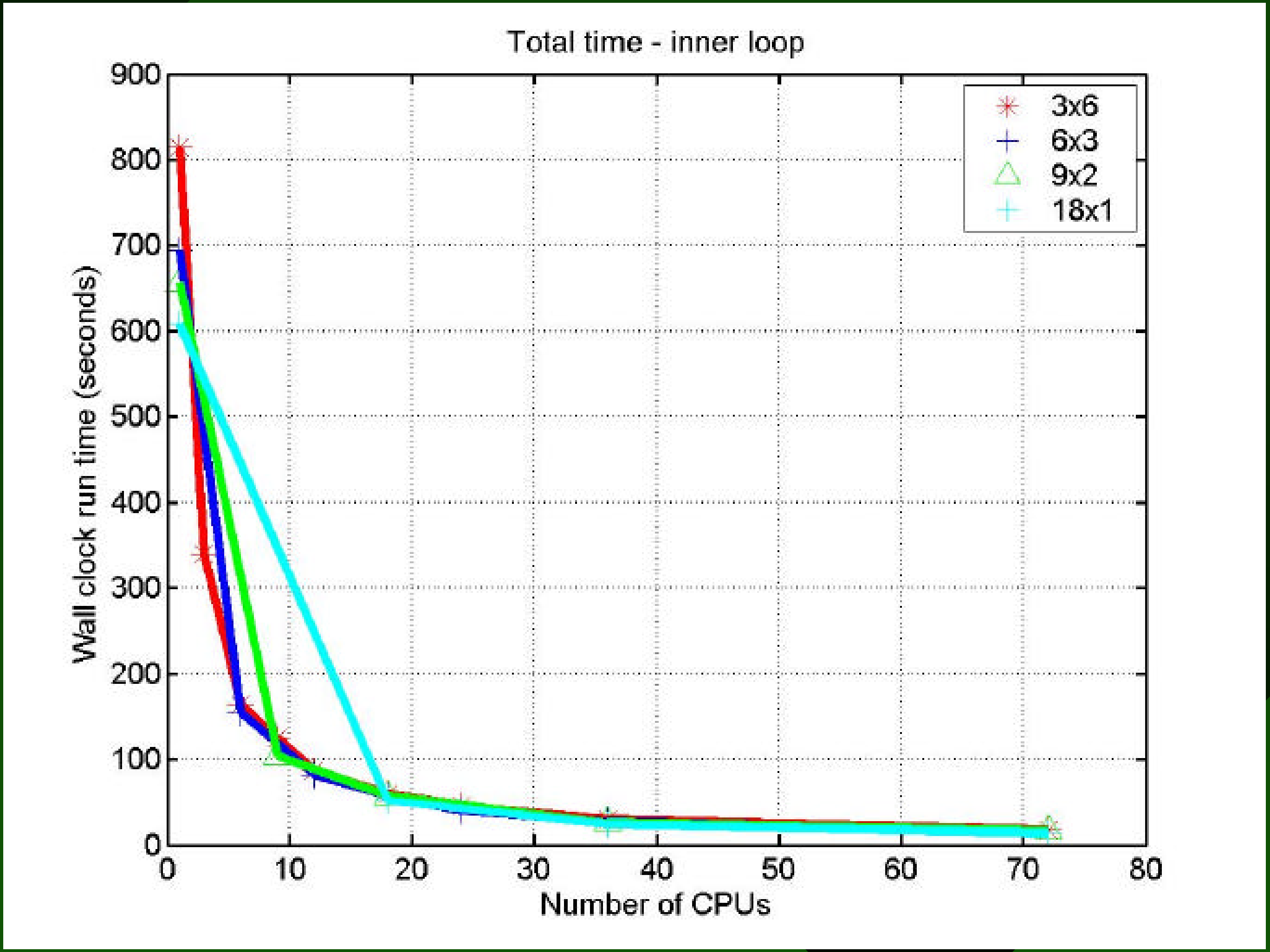
Scalability “M code”



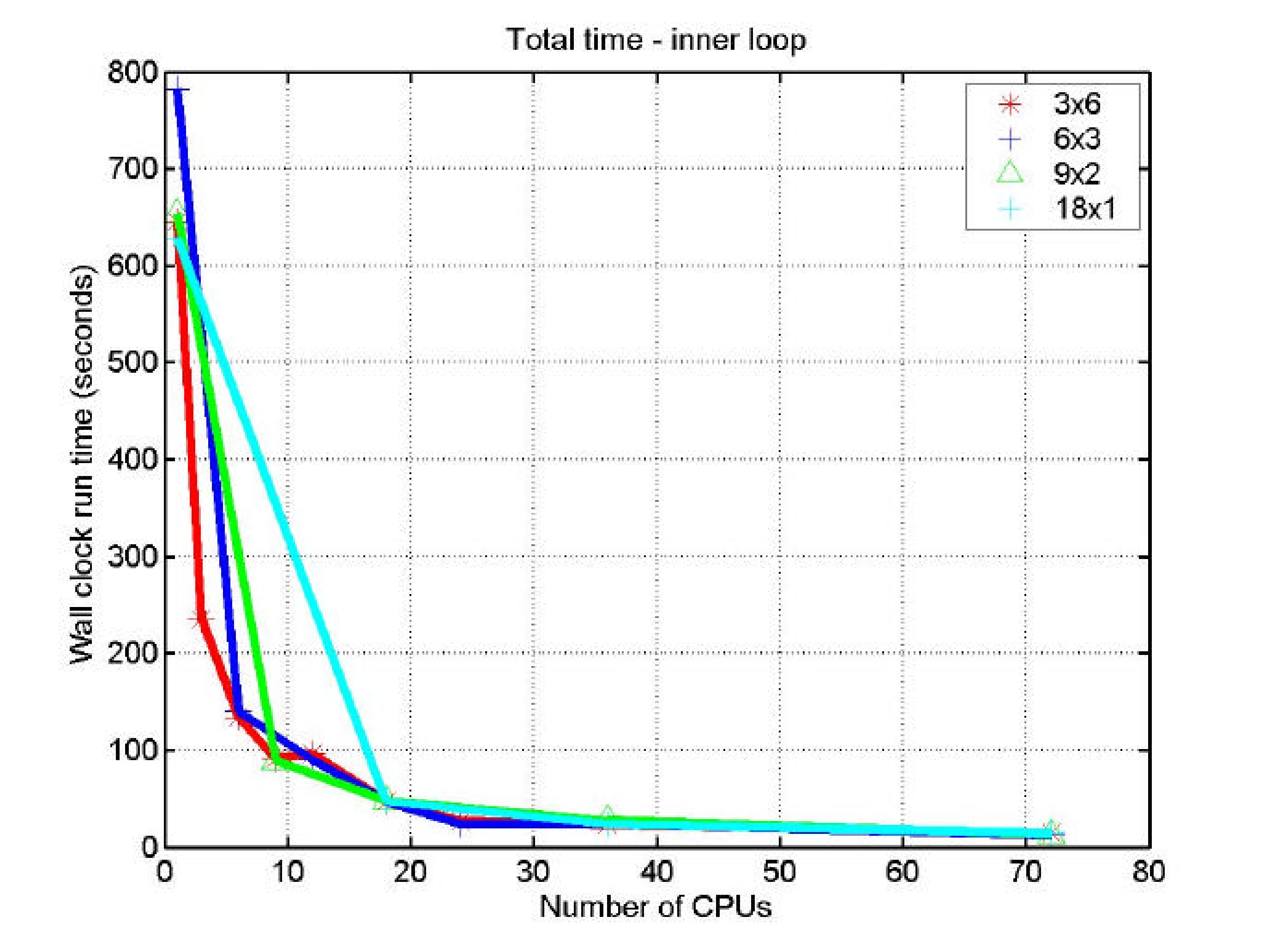
Scalability “C code”



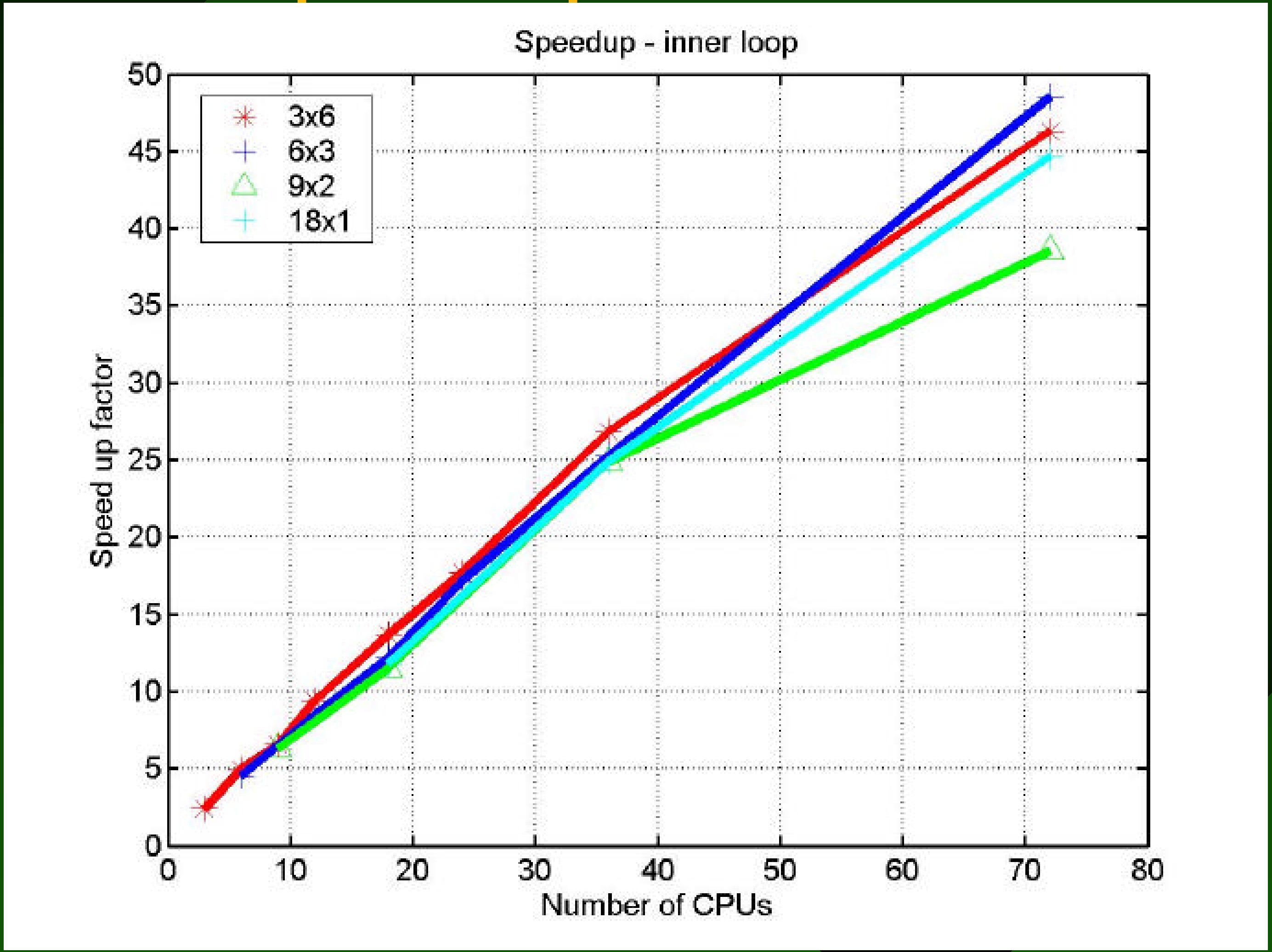
Inner loop time “M code”



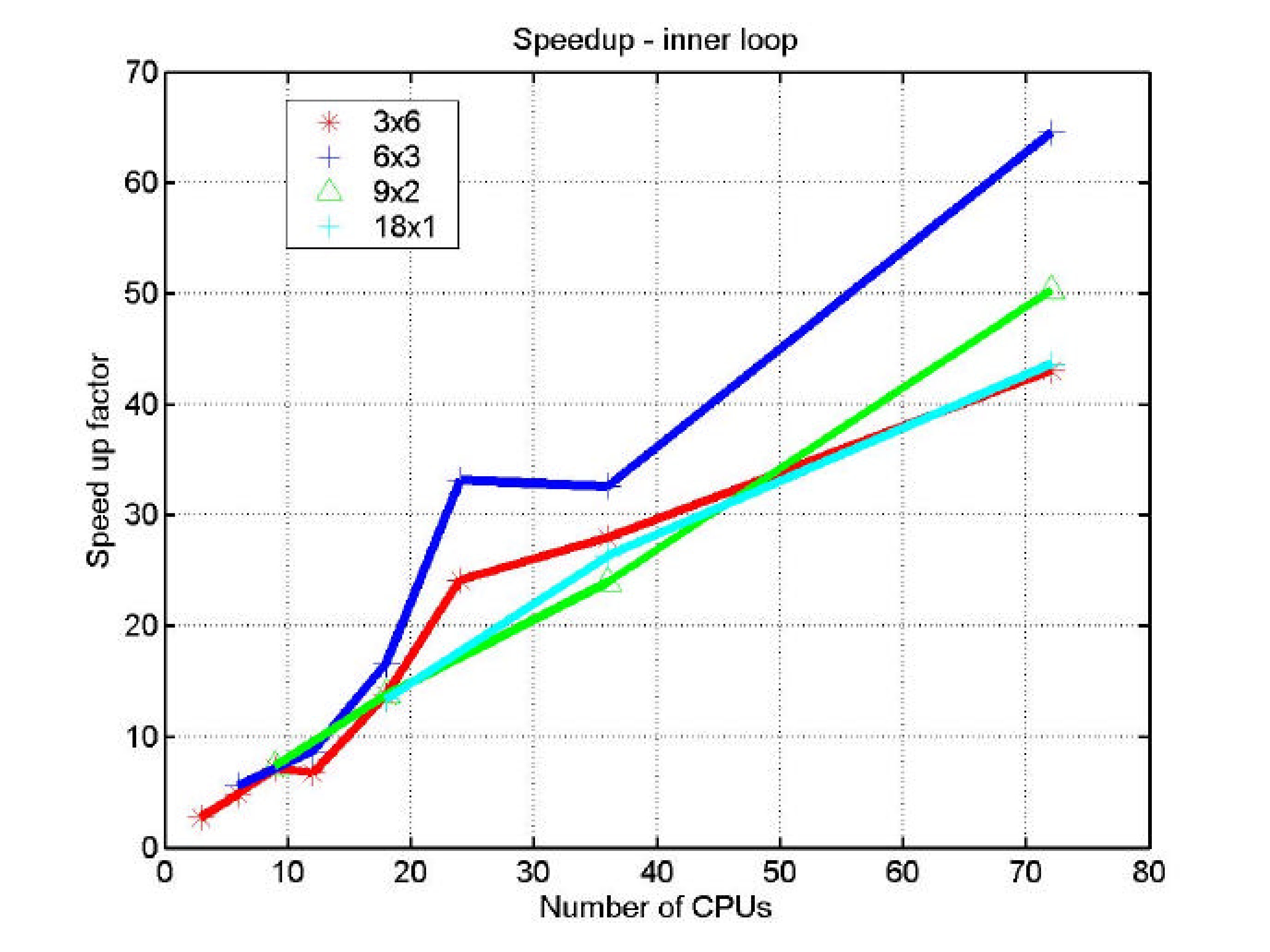
Inner loop time “C code”



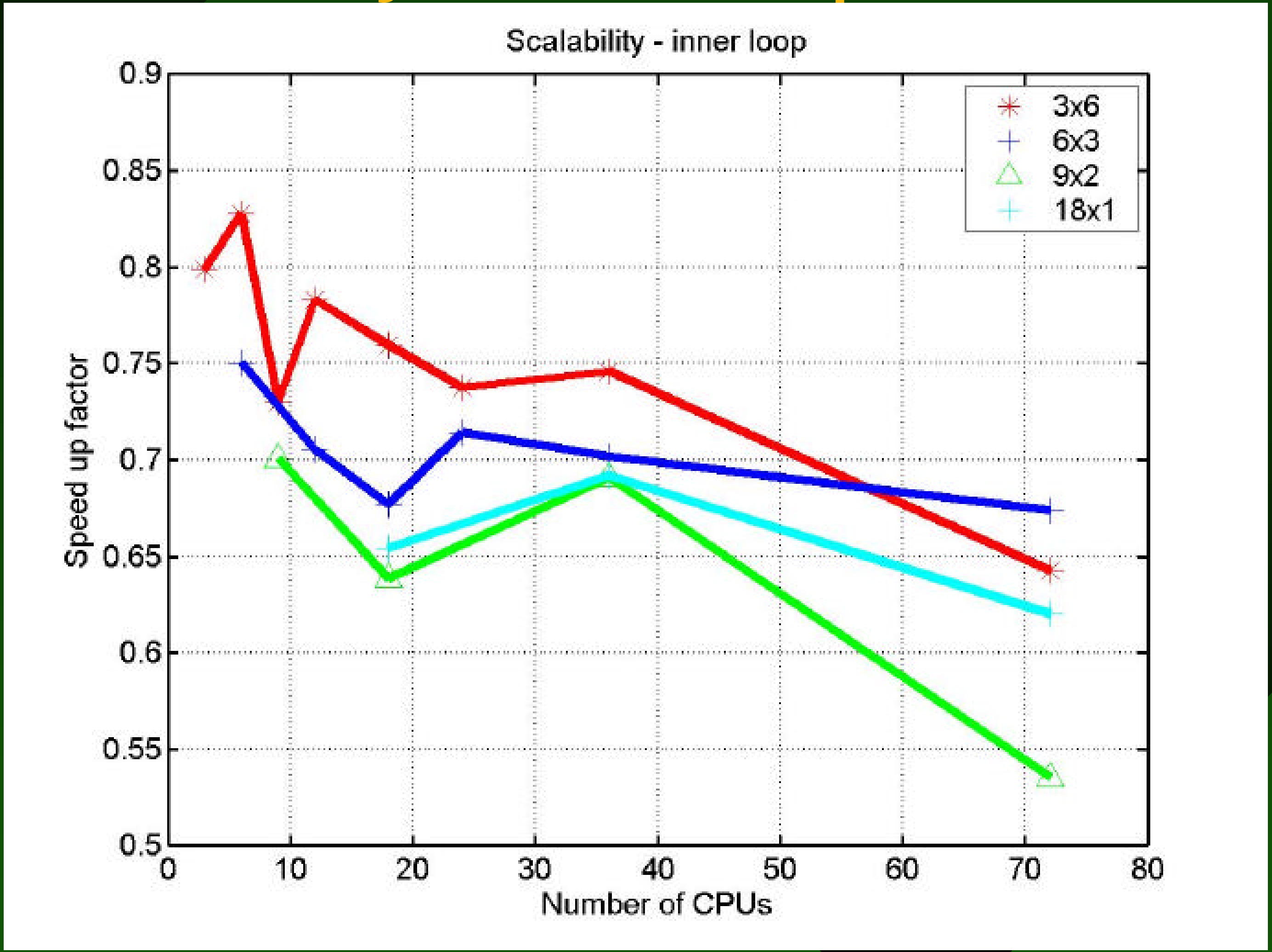
Speedup “M code”



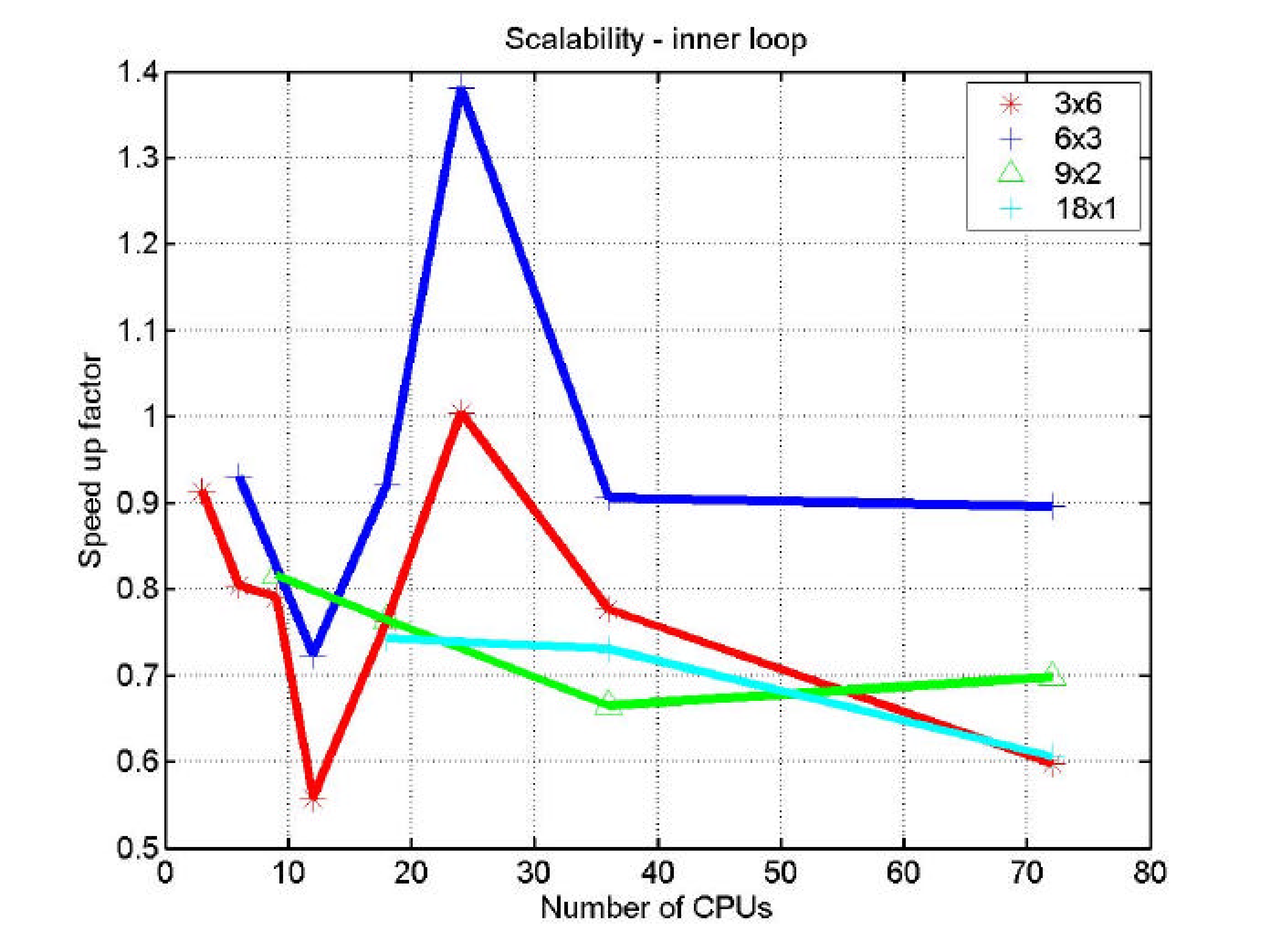
Speedup “C code”



Scalabililty-inner loop “M code”



Scalabililty-inner loop “C code”



Communication time

CPUs type seconds

72 3x6 = 48.9

36 3x6 = 4.35

72 6x3 = 6.09

36 6x3 = 4.71

Less than 0.3 seconds

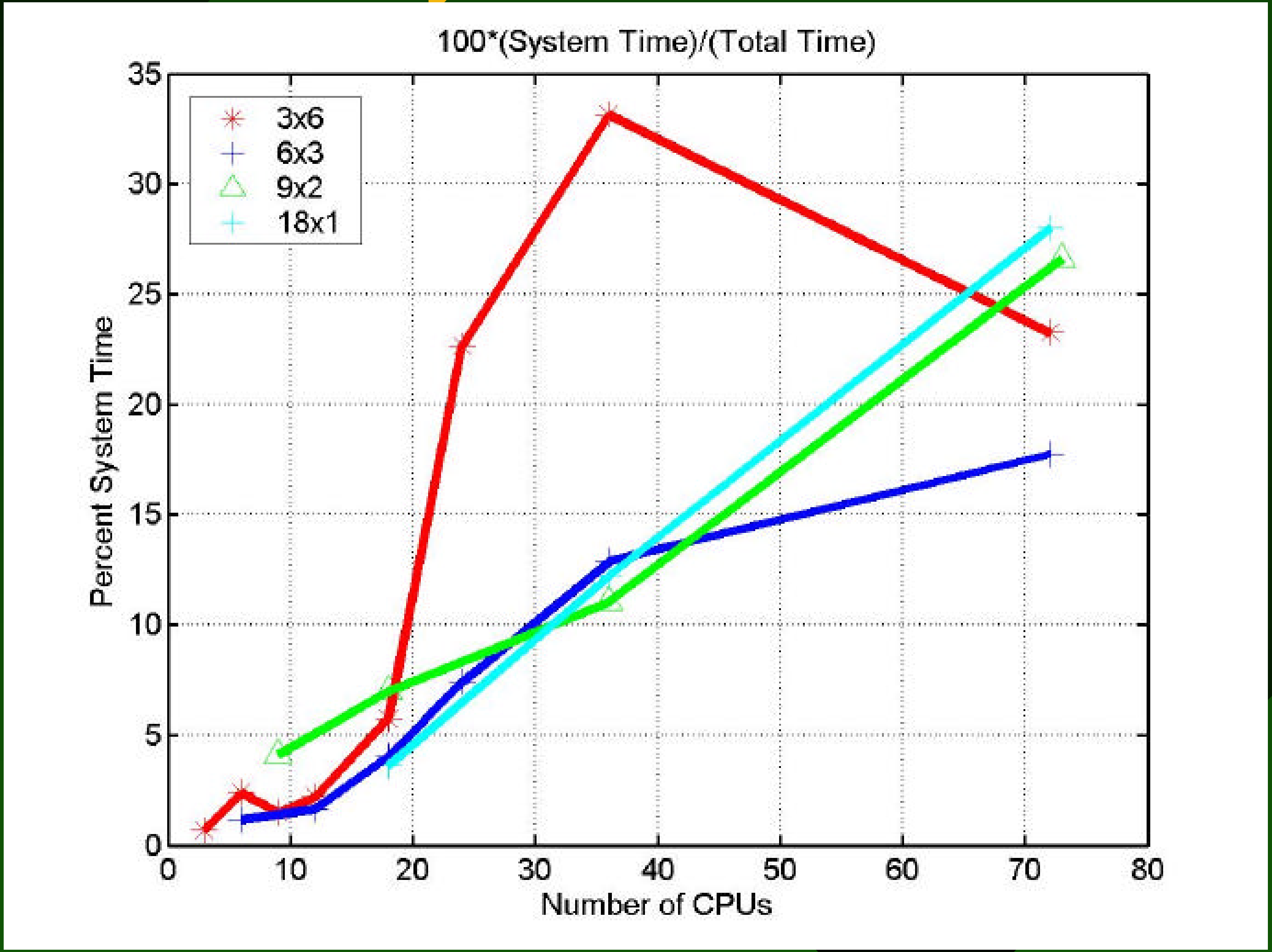
24 18 12 9 6 3 1: 3x6

24 18 12 6 1: 6x3

72 36 18 9 1: 9x2

72 36 18 1: 18x1

System time


100*(System Time)/(Total Time)
Percent System Time
Number of CPUs
0
5
10
15
20
25
30
35
0
10
20
30
40
50
60
70
80
3x6
6x3
9x2
18x1

Summary

- “M-file” MatlabMPI code has comparable performance to compiled “C-code”
- Both methods can be submitted to batch
- “C-code” implementations allow an increase in processor use without the purchase of additional licenses.
- MatlabMPI scales well, but can be influenced when large file transfers are occurring.

Future Work and activities

- Automate the MatlabMPI suite for executable versions.
- Customize MatlabMPI to create batch scripts for all the HPCMP resources
- Matlab via the web – see
  "A Java-based Web Interface to MATLAB"
- Application to BackProjection & Wavefront theory codes.
- HPCMO SIP Benchmark / Profiling Study